可选链式调用（Optional Chaining）

> 1.0 翻译：Jasonbroker (https://github.com/Jasonbroker) 校对：numbbbbb (https://github.com/numbbbbb), stanzhai (https://github.com/stanzhai)
>
> 2.0 翻译+校对：lyojo (https://github.com/lyojo)
>
> 2.1 校对：shanks (http://codebuild.me)，2015-10-31
>
> 2.2 翻译+校对：SketchK (https://github.com/SketchK) 2016-05-15

本页包含内容：

可选链式调用（Optional Chaining）是一种可以在当前值可能为 `nil` 的可选值上请求和调用属性、方法及下标的方法。如果可选值有值，那么调用就会成功；如果可选值是 `nil`，那么调用将返回 `nil`。多个调用可以连接在一起形成一个调用链，如果其中任何一个节点为 `nil`，整个调用链都会失败，即返回 `nil`。

> 注意
> Swift 的可选链式调用和 Objective-C 中向 `nil` 发送消息有些相像，但是 Swift 的可选链式调用可以应用于任意类型，并且能检查调用是否成功。

使用可选链式调用代替强制展开

通过在想调用的属性、方法、或下标的可选值（optional value）后面放一个问号（`?`），可以定义一个可选链。这一点很像在可选值后面放一个叹号（`!`）来强制展开它的值。它们的主要区别在于当可选值为空时可选链式调用只会调用失败，然而强制展开将会触发运行时错误。

为了反映可选链式调用可以在空值（`nil`）上调用的事实，不论这个调用的属性、方法及下标返回的值是不是可选值，它的返回结果都是一个可选值。你可以利用这个返回值来判断你的可选链式调用是否调用成功，如果调用有返回值则说明调用成功，返回 `nil` 则说明调用失败。

特别地，可选链式调用的返回结果与原本的返回结果具有相同的类型，但是被包装成了一个可选值。例如，使用可选链式调用访问属性，当可选链式调用成功时，如果属性原本的返回结果是 `Int` 类型，则会变为 `Int?` 类型。

下面几段代码将解释可选链式调用和强制展开的不同。

首先定义两个类 `Person` 和 `Residence`：

```
class Person {
    var residence: Residence?
}

class Residence {
    var numberOfRooms = 1
}
```

`Residence` 有一个 `Int` 类型的属性 `numberOfRooms`，其默认值为 `1`。`Person` 具有一个可选的 `residence` 属性，其类型为 `Residence?`。

假如你创建了一个新的 `Person` 实例,它的 `residence` 属性由于是是可选型而将初始化为 `nil`,在下面的代码中, `john` 有一个值为 `nil` 的 `residence` 属性：

```
let john = Person()
```

如果使用叹号（`!`）强制展开获得这个 `john` 的 `residence` 属性中的 `numberOfRooms` 值，会触发运行时错误，因为这时 `residence` 没有可以展开的值：

```
let roomCount = john.residence!.numberOfRooms
// 这会引发运行时错误
```

`john.residence` 为非 `nil` 值的时候，上面的调用会成功，并且把 `roomCount` 设置为 `Int` 类型的房间数量。正如上面提到的，当 `residence` 为 `nil` 的时候上面这段代码会触发运行时错误。

可选链式调用提供了另一种访问 `numberOfRooms` 的方式，使用问号（ `?` ）来替代原来的叹号（ `!` ）：

```
if let roomCount = john.residence?.numberOfRooms {
    print("John's residence has \(roomCount) room(s).")
} else {
    print("Unable to retrieve the number of rooms.")
}
// 打印 “Unable to retrieve the number of rooms.”
```

在 `residence` 后面添加问号之后，Swift 就会在 `residence` 不为 `nil` 的情况下访问 `numberOfRooms` 。

因为访问 `numberOfRooms` 有可能失败，可选链式调用会返回 `Int?` 类型，或称为“可选的 `Int` ”。如上例所示，当 `residence` 为 `nil` 的时候，可选的 `Int` 将会为 `nil` ，表明无法访问 `numberOfRooms` 。访问成功时，可选的 `Int` 值会通过可选绑定展开，并赋值给非可选类型的 `roomCount` 常量。

要注意的是，即使 `numberOfRooms` 是非可选的 `Int` 时，这一点也成立。只要使用可选链式调用就意味着 `numberOfRooms` 会返回一个 `Int?` 而不是 `Int` 。

可以将一个 `Residence` 的实例赋给 `john.residence` ，这样它就不再是 `nil` 了：

```
john.residence = Residence()
```

`john.residence` 现在包含一个实际的 `Residence` 实例，而不再是 `nil` 。如果你试图使用先前的可选链式调用访问 `numberOfRooms` ，它现在将返回值为 `1` 的 `Int?` 类型的值：

```
if let roomCount = john.residence?.numberOfRooms {
    print("John's residence has \(roomCount) room(s).")
} else {
    print("Unable to retrieve the number of rooms.")
}
// 打印 “John's residence has 1 room(s).”
```

为可选链式调用定义模型类

通过使用可选链式调用可以调用多层属性、方法和下标。这样可以在复杂的模型中向下访问各种子属性，并且判断能否访问子属性的属性、方法或下标。

下面这段代码定义了四个模型类，这些例子包括多层可选链式调用。为了方便说明，在 `Person` 和 `Residence` 的基础上增加了 `Room` 类和 `Address` 类，以及相关的属性、方法以及下标。

`Person` 类的定义基本保持不变：

```
class Person {
    var residence: Residence?
}
```

`Residence` 类比之前复杂些，增加了一个名为 `rooms` 的变量属性，该属性被初始化为 `[Room]` 类型的空数组：

```
class Residence {
    var rooms = [Room]()
    var numberOfRooms: Int {
        return rooms.count
    }
    subscript(i: Int) -> Room {
        get {
            return rooms[i]
        }
        set {
            rooms[i] = newValue
        }
    }
    func printNumberOfRooms() {
        print("The number of rooms is \(numberOfRooms)")
    }
    var address: Address?
}
```

现在 Residence 有了一个存储 Room 实例的数组， numberOfRooms 属性被实现为计算型属性，而不是存储型属性。 numberOfRooms 属性简单地返回 rooms 数组的 count 属性的值。

Residence 还提供了访问 rooms 数组的快捷方式，即提供可读写的下标来访问 rooms 数组中指定位置的元素。

此外， Residence 还提供了 printNumberOfRooms() 方法，这个方法的作用是打印 numberOfRooms 的值。

最后， Residence 还定义了一个可选属性 address ，其类型为 Address? 。 Address 类的定义在下面会说明。

Room 类是一个简单类，其实例被存储在 rooms 数组中。该类只包含一个属性 name ，以及一个用于将该属性设置为适当的房间名的初始化函数：

```
class Room {
    let name: String
    init(name: String) { self.name = name }
}
```

最后一个类是 Address ，这个类有三个 String? 类型的可选属性。 buildingName 以及 buildingNumber 属性分别表示某个大厦的名称和号码，第三个属性 street 表示大厦所在街道的名称：

```
class Address {
    var buildingName: String?
    var buildingNumber: String?
    var street: String?
    func buildingIdentifier() -> String? {
        if buildingName != nil {
            return buildingName
        } else if buildingNumber != nil && street != nil {
            return "\(buildingNumber) \(street)"
        } else {
            return nil
        }
    }
}
```

Address 类提供了 buildingIdentifier() 方法，返回值为 String? 。 如果 buildingName 有值则返回 buildingName 。或者，如果 buildingNumber 和 street 均有值则返回 buildingNumber 。否则，返回 nil 。

通过可选链式调用访问属性

正如使用可选链式调用代替强制展开中所述，可以通过可选链式调用在一个可选值上访问它的属性，并判断访问是否成功。

下面的代码创建了一个 Person 实例，然后像之前一样，尝试访问 numberOfRooms 属性：

```
let john = Person()
if let roomCount = john.residence?.numberOfRooms {
    print("John's residence has \(roomCount) room(s).")
} else {
    print("Unable to retrieve the number of rooms.")
}
// 打印 “Unable to retrieve the number of rooms.”
```

因为 john.residence 为 nil ，所以这个可选链式调用依旧会像先前一样失败。

还可以通过可选链式调用来设置属性值：

```
let someAddress = Address()
someAddress.buildingNumber = "29"
someAddress.street = "Acacia Road"
john.residence?.address = someAddress
```

在这个例子中，通过 john.residence 来设定 address 属性也会失败，因为 john.residence 当前为 nil 。

上面代码中的赋值过程是可选链式调用的一部分，这意味着可选链式调用失败时，等号右侧的代码不会被执行。对于上面的代码来说，很难验证这一点，因为像这样赋值一个常量没有任何副作用。下面的代码完成了同样的事情，但是它使用一个函数来创建 Address 实例，然后将该实例返回用于赋值。该函数会在返回前打印“Function was called”，这使你能验证等号右侧的代码是否被执行。

```
func createAddress() -> Address {
    print("Function was called.")

    let someAddress = Address()
    someAddress.buildingNumber = "29"
    someAddress.street = "Acacia Road"

    return someAddress
}
john.residence?.address = createAddress()
```

没有任何打印消息，可以看出 `createAddress()` 函数并未被执行。

通过可选链式调用调用方法

可以通过可选链式调用来调用方法，并判断是否调用成功，即使这个方法没有返回值。

`Residence` 类中的 `printNumberOfRooms()` 方法打印当前的 `numberOfRooms` 值，如下所示：

```
func printNumberOfRooms() {
    print("The number of rooms is \(numberOfRooms)")
}
```

这个方法没有返回值。然而，没有返回值的方法具有隐式的返回类型 `Void` ，如无返回值函数(./06_Functions.html#functions_without_return_values)中所述。这意味着没有返回值的方法也会返回 `()` ，或者说空的元组。

如果在可选值上通过可选链式调用来调用这个方法，该方法的返回类型会是 `Void?` ，而不是 `Void` ，因为通过可选链式调用得到的返回值都是可选的。这样我们就可以使用 `if` 语句来判断能否成功调用 `printNumberOfRooms()` 方法，即使方法本身没有定义返回值。通过判断返回值是否为 `nil` 可以判断调用是否成功：

```
if john.residence?.printNumberOfRooms() != nil {
    print("It was possible to print the number of rooms.")
} else {
    print("It was not possible to print the number of rooms.")
}
// 打印 “It was not possible to print the number of rooms.”
```

同样的，可以据此判断通过可选链式调用为属性赋值是否成功。在上面的通过可选链式调用访问属性的例子中，我们尝试给 `john.residence` 中的 `address` 属性赋值，即使 `residence` 为 `nil` 。通过可选链式调用给属性赋值会返回 `Void?` ，通过判断返回值是否为 `nil` 就可以知道赋值是否成功：

```
if (john.residence?.address = someAddress) != nil {
    print("It was possible to set the address.")
} else {
    print("It was not possible to set the address.")
}
// 打印 “It was not possible to set the address.”
```

通过可选链式调用访问下标

通过可选链式调用，我们可以在一个可选值上访问下标，并且判断下标调用是否成功。

> 注意
> 通过可选链式调用访问可选值的下标时，应该将问号放在下标方括号的前面而不是后面。可选链式调用的问号一般直接跟在可选表达式的后面。

下面这个例子用下标访问 `john.residence` 属性存储的 `Residence` 实例的 `rooms` 数组中的第一个房间的名称，因为 `john.residence` 为 `nil` ，所以下标调用失败了：

```
if let firstRoomName = john.residence?[0].name {
    print("The first room name is \(firstRoomName).")
} else {
    print("Unable to retrieve the first room name.")
}
// 打印 “Unable to retrieve the first room name.”
```

在这个例子中，问号直接放在 `john.residence` 的后面，并且在方括号的前面，因为 `john.residence` 是可选值。

类似的，可以通过下标，用可选链式调用来赋值：

```
john.residence?[0] = Room(name: "Bathroom")
```

这次赋值同样会失败，因为 `residence` 目前是 `nil` 。

如果你创建一个 `Residence` 实例，并为其 `rooms` 数组添加一些 `Room` 实例，然后将 `Residence` 实例赋值给 `john.residence` ，那就可以通过可选链和下标来访问数组中的元素：

```
let johnsHouse = Residence()
johnsHouse.rooms.append(Room(name: "Living Room"))
johnsHouse.rooms.append(Room(name: "Kitchen"))
john.residence = johnsHouse

if let firstRoomName = john.residence?[0].name {
    print("The first room name is \(firstRoomName).")
} else {
    print("Unable to retrieve the first room name.")
}
// 打印 “The first room name is Living Room.”
```

访问可选类型的下标

如果下标返回可选类型值，比如 Swift 中 `Dictionary` 类型的键的下标，可以在下标的结尾括号后面放一个问号来在其可选返回值上进行可选链式调用：

```
var testScores = ["Dave": [86, 82, 84], "Bev": [79, 94, 81]]
testScores["Dave"]?[0] = 91
testScores["Bev"]?[0] += 1
testScores["Brian"]?[0] = 72
// "Dave" 数组现在是 [91, 82, 84]，"Bev" 数组现在是 [80, 94, 81]
```

上面的例子中定义了一个 `testScores` 数组，包含了两个键值对，把 `String` 类型的键映射到一个 `Int` 值的数组。这个例子用可选链式调用把 `"Dave"` 数组中第一个元素设为 `91` ，把 `"Bev"` 数组的第一个元素 `+1` ，然后尝试把 `"Brian"` 数组中的第一个元素设为 `72` 。前两个调用成功，因为 `testScores` 字典中包含 `"Dave"` 和 `"Bev"` 这两个键。但是 `testScores` 字典中没有 `"Brian"` 这个键，所以第三个调用失败。

连接多层可选链式调用

可以通过连接多个可选链式调用在更深的模型层级中访问属性、方法以及下标。然而，多层可选链式调用不会增加返回值的可选层级。

也就是说：

- 如果你访问的值不是可选的，可选链式调用将会返回可选值。
- 如果你访问的值就是可选的，可选链式调用不会让可选返回值变得“更可选”。

因此：

- 通过可选链式调用访问一个 `Int` 值，将会返回 `Int?` ，无论使用了多少层可选链式调用。
- 类似的，通过可选链式调用访问 `Int?` 值，依旧会返回 `Int?` 值，并不会返回 `Int??` 。

下面的例子尝试访问 `john` 中的 `residence` 属性中的 `address` 属性中的 `street` 属性。这里使用了两层可选链式调用， `residence` 以及 `address` 都是可选值：

```
if let johnsStreet = john.residence?.address?.street {
    print("John's street name is \(johnsStreet).")
} else {
    print("Unable to retrieve the address.")
}
// 打印 “Unable to retrieve the address.”
```

`john.residence` 现在包含一个有效的 `Residence` 实例。然而， `john.residence.address` 的值当前为 `nil` 。因此，调用 `john.residence?.address?.street` 会失败。

需要注意的是，上面的例子中， `street` 的属性为 `String?` 。 `john.residence?.address?.street` 的返回值也依然是 `String?` ，即使已经使用了两层可选链式调用。

如果为 `john.residence.address` 赋值一个 `Address` 实例，并且为 `address` 中的 `street` 属性设置一个有效值，我们就能过通过可选链式调用来访问 `street` 属性：

```
let johnsAddress = Address()
johnsAddress.buildingName = "The Larches"
johnsAddress.street = "Laurel Street"
john.residence?.address = johnsAddress

if let johnsStreet = john.residence?.address?.street {
    print("John's street name is \(johnsStreet).")
} else {
    print("Unable to retrieve the address.")
}
// 打印 “John's street name is Laurel Street.”
```

在上面的例子中，因为 `john.residence` 包含一个有效的 `Residence` 实例，所以对 `john.residence` 的 `address` 属性赋值将会成功。

在方法的可选返回值上进行可选链式调用

上面的例子展示了如何在一个可选值上通过可选链式调用来获取它的属性值。我们还可以在一个可选值上通过可选链式调用来调用方法，并且可以根据需要继续在方法的可选返回值上进行可选链式调用。

在下面的例子中，通过可选链式调用来调用 `Address` 的 `buildingIdentifier()` 方法。这个方法返回 `String?` 类型的值。如上所述，通过可选链式调用来调用该方法，最终的返回值依旧会是 `String?` 类型：

```
if let buildingIdentifier = john.residence?.address?.buildingIdentifier() {
    print("John's building identifier is \(buildingIdentifier).")
}
// 打印 “John's building identifier is The Larches.”
```

如果要在该方法的返回值上进行可选链式调用，在方法的圆括号后面加上问号即可：

```
if let beginsWithThe =
    john.residence?.address?.buildingIdentifier()?.hasPrefix("The") {
        if beginsWithThe {
            print("John's building identifier begins with \"The\".")
        } else {
            print("John's building identifier does not begin with \"The\".")
        }
}
// 打印 “John's building identifier begins with "The".”
```

> 注意
> 在上面的例子中，在方法的圆括号后面加上问号是因为你要在 `buildingIdentifier()` 方法的可选返回值上进行可选链式调用，而不是方法本身。

上一篇: 自动引用计数 (/project/swift/chapter2/16_Automatic_Reference_Counting.html)
下一篇: 错误处理 (/project/swift/chapter2/18_Error_Handling.html)

被顶起来的评论

TRY美剧 (http://weibo.com/dejavuzhou)
hot hot hot!!! - Randy (South Park)
(http://weibo.com/dejavuzhou)2015年11月10日 回复 顶(1) 转发

11 条评论 1 条新浪微博 最新 最早 最热

Taylor3
如果专门拿出一章来介绍可选类的用法就好了。
6月2日 回复 顶 转发

RiversMa (http://weibo.com/5540013595)
回复 水木十里: 有这个方法的
(http://weibo.com/5540013595)5月31日 回复 顶 转发

卡耐
为什么，retrun buildingName 不是可选，而如果改为 "\(buildingName)" 就是可选了?
5月28日 回复 顶 转发

outhur
最后一个例子，hasPrefix("The")，我和他书上说的不一样，我是得到 Optional("The Larches.")，所以，前缀没有 “The”，所以最后打印的是 “John's building identifier does not begin with “The”.”
5月28日 回复 顶 转发